Santa Clara University School of Law

Santa Clara High Technology Law Journal

Volume 21 | Issue 4 Article 4

2005

No More Soft Landings for Software: Liability for Defects in an Industry That Has Come of Age

Frances E. Zollers

Andrew McMullin

Sandra N. Hurd

Peter Shears

NO MORE SOFT LANDINGS FOR SOFTWARE: LIABILITY FOR DEFECTS IN AN INDUSTRY THAT HAS COME OF AGE

Frances E. Zollers, Andrew McMullin, Sandra N. Hurd, and Peter Shears†

I. INTRODUCTION

This is not a tale of robots gone wild or other stuff of science fiction. Rather, this is about real-life situations that have occurred or are likely to occur. It is about software failure when that failure leads, not to system crashes or botched tax returns, but to serious physical injury to persons. The power of software can be seen everywhere: It flies airplanes, monitors medical patients and nuclear power plants, and even helps us drive our cars. Indeed, software is no longer confined to the domain of business systems that control inventory, issue payroll checks, and keep track of accounts receivable and payable. It extends beyond the desktop computer with its word processing and data management capabilities and now routinely interfaces with human beings in their daily lives and in unseen ways. The consequence, however, is that some software can cause physical injury if it is defective. While many early commentators have speculated about the liability regime when such a condition occurs,[1] it

† Frances E. Zollers is Professor of Law & Public Policy at the Whitman School of Management, Syracuse University. Andrew McMullin is Technical Manager, Faculty of Social Science & Business, University of Plymouth, Plymouth, England (e-mail: andy.mcmullin@plymouth.ac.uk). Sandra N. Hurd is Professor of Law & Public Policy at the Whitman School of Management, Syracuse University (snhurd@syr.edu). Peter Shears is Director of Professional Studies at the School of Sociology, Politics & Law, University of Plymouth, Plymouth, England (peter.shears@plymouth.ac.uk). Correspondence should be directed to Frances E. Zollers, Professor of Law & Public Policy, Whitman School of Management, Syracuse University, Syracuse, NY 13244-2450; Tel.: 315- 443-3648; Fax: 315-443-2185; e-mail: fzollers@syr.edu. The authors thank the Earl V. Snyder Center for Innovation Management at the Whitman School of Management, Syracuse University, for research support and Quyen Luu for her research assistance.

1. Vincent M. Brannigan & Ruth E. Dayhoff, *Liability for Personal Injuries Caused by Defective Medical Computer Programs*, 7 AM. J.L. & MED. 123 (1981); Bruce Ducker, *Liability for Computer Software*, 26 BUS. LAW. 1081 (1971); Roy N. Freed, *Legal Questions in a Computer Society*, TRIAL, Jan./Feb. 1971, at 39 [hereinafter Freed, *Legal Questions*]; Roy N.

is now time to take stock of how the law is developing and should develop when software foreseeably causes physical injury.

The discussion is timely for a number of reasons. First, there have been sufficient numbers of instances of software failure that have caused physical injury[2] to cause serious concern, and the number can only grow, given the pervasiveness of software in our daily lives. Second, the software industry is no longer in its infancy. Its development has moved out of garages and into corporate offices. It has matured to become a dominant sector of the economy. Consequently, it is appropriate to consider liability for defective software in the same light as liability for defective automobiles, pharmaceuticals, and other products.

This article will first examine the characteristics of software and its evolutionary creep into our lives in Part II. In Parts III and IV, we will review the literature about software litigation and the eras through which it has progressed. Part V reviews the origins of strict product liability and the policies underlying it to determine whether software, which has hitherto enjoyed immunity from strict liability, fits into the strict liability context. Lastly, in Part VI we argue for the adoption of a strict liability regime for software failure that produces physical injury and offer supporting arguments for why such a move is both necessary and sensible.

II. COMPUTER HISTORY 101

While many think that computers, software, and computing are relatively new developments, this is not really the case. A timeline showing the major events in the history of the development of the computer should really start with Napier's bones in the 17th Century and include such developments as Pascal's adding machine,

---

Freed, *Products Liability in the Computer Age*, 17 JURIMETRICS J. 270 (1977) [hereinafter Freed, *Computer Age*]; Roy N. Freed, *The Effect of Computer Technology on Legal Liability* (1962) (proceedings of Wisconsin Eighth Annual Corporate Lawyer's Institute), *in* ROY N. FREED, COMPUTERS AND LAW – A REFERENCE WORK 24 (4th ed. 1974) [hereinafter Freed, *Computer Technology*]; Michael C. Gemignani, *Product Liability and Software*, 8 RUTGERS COMPUTER & TECH. L.J. 173 (1981); David A. Hall, Note, *Strict Products Liability and Computer Software: Caveat Vendor*, 4 COMPUTER/L.J. 373 (1983); Bonna Lynn Horowitz, Note, *Computer Software as a Good Under the Uniform Commercial Code: Taking a Byte Out of the Intangibility Myth*, 65 B.U. L. REV. 129 (1985); Susan Lanoue, Comment, *Computer Software and Strict Products Liability*, 20 SAN DIEGO L. REV. 439 (1983).

2. Gemignani, *supra* note 1; Thomas G. Wolpert, *Product Liability and Software Implicated in Personal Injury*, 60 DEF. COUNS. J. 519 (1993).

Jacquard's loom, and Herman Hollerith's tabulating code developed for the American census of 1890.[3]

Although the history of computers and computing becomes clouded by the security surrounding both the Second World War and the Cold War that followed, two major developments in 1943 are generally regarded as the origination of the modern stored program electronic computer: J. Prosper Eckert and Dr. John W. Mauchly's (United States) ENIAC[4] design and the design at Bletchley Park (England)[5] of Colossus, the computer that would decode the German enigma messages.[6]

Alan Turing, sometimes called the father of computer science, tied together these two developments, and it was he who considered the fundamentals of what is now called software fault tolerance.[7] Turing concluded that it was not possible to write a program that could determine if another program will compute successfully and halt.[8] In other words, the proof of correctness of any program is not computable, and software fault tolerance does not have a theoretical basis.

Neither ENIAC nor Colossus were true programmable computers in the modern sense. The development of magnetic core computer memories (patented in 1947) led to the move away from programming done by wiring a physical connection between the parts of the hardware, towards programming done by storing an easily altered program in the machine's memory.

The development of the transistor at Bell labs in the late 1940s as a replacement for unreliable relays[9] started the next phase of expansion of computers and software into our lives. Before then,

---

3. The Computer Museum at Bletchley Park, *Timeline*, *at* http://www.retrobeep.com/timeline/timelineIndex.htm (last visited July 14, 2004).

4. Martin H Weik, *The ENIAC Story*, ORDINANCE, Jan.-Feb. 1961, *available at* http://ftp.arl.army.mil/~mike/comphist/eniac-story.html (last visited July 14, 2004).

5. Bletchley Park, Official Homepage, *at* http://www.bletchleypark.org.uk/ (last visited July 14, 2004).

6. Tony Sale, *The Colossus Rebuild Project*, *at* http://www.codesandciphers.org.uk/lorenz/rebuild.htm (last visited July 14, 2004).

7. Marathon Techs. Corp., *The History of Fault Tolerance Explained*, Real Time Sys. Ltd. (1998), *at* http://www.rts2000.demon.co.uk/historyoffaulttolerance.html (last visited July 14, 2004).

8. *Id.*

9. Lucent Techs., *The Transistor—First Uses* (2004), *at* http://www.lucent.com/minds/transistor/uses1.html (last visited Mar. 8, 2005); Lucent Techs., *The Transistor—The Inventors* (2004), *at* http://www.lucent.com/minds/transistor/inventors0.html (last visited Mar. 8, 2005).

electronics mainly relied upon valves[10] that were used in the logic circuits of the early computers. Valves require a lot of power, generate a lot of heat when working correctly, and are fragile. Transistors,[11] on the other hand, require less power, generate less heat and are more reliable.

It was the work of Jack Kilby[12] in the late 1950s when working for Texas Instruments that led to the idea of creating other components out of the materials used to construct transistors and making them into a single package. Thus was born the concept of the integrated circuit from which Intel in 1969 created the first microprocessor.[13]

Since the design of the first microprocessor, subsequent designs have primarily increased system speeds and system capability while decreasing costs. It was this growth in power and decrease in costs that led directly to the microprocessor becoming ubiquitous, powering an estimated 1,000,000,000 home computers in 2002.[14] Microprocessors are also found in washing machines and dishwashers (replacing the previously common clockwork timing mechanisms) and are found controlling phase-locked loops tuning portable radios, replacing the previous analogue designs of capacitors and inductors. Microprocessors with allied circuits have replaced the mechanical tape mechanism in the answering machine and the mechanical dial of the telephone. Microprocessors control the engine management, braking system, airbags, navigation system, radio, cruise control, four-wheel drive, and even the wiring of cars from all over the world.[15]

The use of a generic microprocessor and special software instead of discrete components wired for a specific purpose has led to a "throw away" attitude with respect to modern electronics. When it ceased to be viable to repair the electronic components within modern

---

10. *See* ScienCentral, Inc. & The Am. Inst. of Physics, *The Vacuum Tube,* PBS Online, *at* http://www.pbs.org/transistor/science/events/vacuumt.html (last visited Mar. 3, 2005).

11. ScienCentral, Inc. & The American Inst. of Physics, *Transistorized!*, PBS Online, *at* http://www.pbs.org/transistor/album1/index.html (last visited Mar. 3, 2005).

12. *See generally Inventor Jack Kilby*, The Great Idea Finder, *at* http://www.ideafinder.com/history/inventors/kilby.htm (last visited Mar. 3, 2005) (short biography).

13. *Microprocessor History—Invention of the Microprocessor*, The Great Idea Finder, *at* http://www.ideafinder.com/history/inventions/microprocessor.htm (last visited Mar. 3, 2005).

14. *Computers Reach One Billion Mark*, BBC NEWS, July 1, 2002, *at* http://news.bbc.co.uk/1/hi/sci/tech/2077986.stm (last visited Mar. 3, 2005).

15. *See, e.g.*, InfoArt News Agency, *Intel Processors Will Power Russian Cars,* InfoArt, *at* http://scripts.infoart.ru/it/news/engnews/99/04/02_405.htm (last visited Mar. 3, 2005).

equipment, it became less expensive to replace the whole item, which led manufacturers to adopt the attitude that "it will be all right in the next revision." This attitude has now spread from hardware to software and might be one of the root causes of failures of some systems.

Because programs can be written to do things that conventional components cannot, microprocessors have helped in the creation of previously impossible new products such as digital cameras, mobile telephones, pocket computers, portable printers, small GPS receivers, and CD/DVD players. However, while all these products seem to be "cutting edge," the processor inside, at the heart of these devices, is still based upon concepts devised over fifty years ago. Thus, the technology contained within the product should be considered mature, not something new.

A. Software Versus Hardware

This history of the computer has concentrated upon the hardware that makes up the machines. Hardware is the "nuts and bolts"—the parts of a computer system that you can see, touch, and feel.[16] Typically hardware either works or it does not; the power supply gives out twelve volts or it does not, the disk drive spins or it does not. Hardware can fail for one of two reasons, faulty design or failed components.[17]

The programs that make the hardware do what the user wants the hardware to do are collectively called the software.[18] Software, supplied to the user by the hardware (disk) on which the software is recorded, is the computer instructions and data that are stored electronically within the computer; when the electricity is turned off, the software ceases to exist.

B. What Is a Software Failure and Why Does Software Fail?

Software can only fail for one reason: faulty design.[19] While it is common to call any failure of software to perform as the user

16. *See Hardware*, MERRIAM-WEBSTER ONLINE DICTIONARY, *at* http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=hardware&x=15&y=15 (last visited Mar. 3, 2005) (definition).

17. Marathon Techs. Corp., *supra* note 7.

18. *See Software*, MERRIAM-WEBSTER ONLINE DICTIONARY, *at* http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=software&x=15&y=15 (last visited Mar. 3, 2005) (definition).

19. Marathon Techs. Corp., *supra* note 7.

expected a bug,[20] the real bug is the failure of the software to do what the programmer who created it thought it was going to do, which is not quite the same thing. Many reported "bugs" are, in fact, the correct and expected operation of the hardware and software combination, but they are not what the end-user expected it to do, so they may be considered faults, but not failures.

Software systems are products of the human mind and are vulnerable to programming mistakes. While software can only fail due to faulty design, these mistakes occur in various forms, including design inconsistencies, syntax errors, and semantic errors.

In everyday desktop systems, these glitches can result in inconvenience and frustration, but are not usually particularly serious. However, even as small an error as a punctuation mistake can cause major economic loss and mission failure as was seen in the North American Space Agency's ("NASA") Mariner 1 Venus probe that launched in 1962, where a missing hyphen caused the spacecraft to travel off-course.[21]

Newer programming techniques and compilers can mitigate many semantic and syntactic errors by increasing the amount of checking done by the compiler over the programming code entered into it. However, as Turing's proof demonstrated, they cannot test the complete correctness of any specific program. Unfortunately, tests cannot prove that there are absolutely no bugs in a program. All they can do is highlight the existence of the problems they find.

A researcher at NASA, Ames M. Lowry, argued in a report that the size of software used in space missions has been increasing exponentially.[22] He noted that there are many examples where bugs in contemporary systems have resulted in mission failure and suggests that the number of software errors in future systems may similarly increase, thereby increasing the likelihood of critical faults, which

---

20. Legend is that in 1946 the term "bug" was first coined to describe an error in the execution of a computer's program when Grace Hopper was working at the computation laboratory at Harvard and traced an error to a moth trapped in a relay. The dead insect was removed from the relay, taped into her logbook, and recorded as the first bug. Rebecca Norman, *Grace Murray Hopper*, *at* http://www.agnesscott.edu/lriddle/women/hopper.htm (last visited Mar. 27, 2005) (short biography).

21. *NSSDC Master Catalog: Spacecraft, Mariner 1*, National Space Science Data Center, *at* http://nssdc.gsfc.nasa.gov/nmc/tmp/MARIN1.html (last visited July 14, 2004).

22. MICHAEL R. LOWRY, SOFTWARE CONSTRUCTION AND ANALYSIS TOOLS FOR FUTURE SPACE MISSIONS 3 (2001), *available at* http://www-etaps.imag.fr/Invited/Lowry-tacas.pdf (last visited April 14, 2005).

may "incapacitate safety-critical systems."[23] In the context of the report, a safety-critical system is defined as one in which a malfunction could result in death, injury or illness, major economic loss, mission failure, environmental damage, or property damage.[24] In the *IEE Review*, Les Hatton asserts that the average number of errors per line of computer code remains fairly constant for different source languages.[25] This is interpreted to mean that the brain of the programmer has a constant probability of introducing faults. In other words, doubling the number of lines of code will produce double the number of programming faults.

C. How Can Software Failure Be Mitigated?

Modern software engineering techniques[26] for creating and maintaining software applications combine techniques from computer science, project management, domain knowledge, and other skills and technologies, including common sense. The practices have evolved steadily, but it was the supposed software crisis[27] during the period from 1960 to 1980 that identified many problems of software development. During that time, projects were delivered late, went over budget, and some caused property damage or loss of life.

To combat the problems, a number of tools, including new ways of working, new processes, and new languages, were all suggested as solutions. However, in his article "No Silver Bullet,"[28] Fred Brooks argues that there can be no easy answers to software engineering problems, describing two different types of complexity. Essential complexity is inherent and nothing can remove it.[29] If a word

---

23. ROBERT DIMOND, ET AL., SURPRISE 2002, SOFTWARE FOR NASA IN 2050: AN IMPOSSIBLE MISSION? 3 (2002), *available at* http://www.matcore.com/surprise/report/Report_FINAL.pdf (last visited Mar. 27, 2005) (citing LOWRY, *supra* note 22).

24. *Id.*

25. Les Hatton, *Software Failures, Follies and Fallacies*, 43 IEE REVIEW, Mar. 1997, at 49.

26. *See Software Engineering*, Wikipedia, *at* http://en.wikipedia.org/wiki/Software_engineering (last visited Mar. 7, 2005) (providing a brief overview of software engineering).

27. *See Software Crisis*, Wikipedia, *at* http://en.wikipedia.org/wiki/Software_crisis (last visited Mar. 7, 2005) (providing a brief overview of the software crisis).

28. Frederick P. Brooks, Jr., *No Silver Bullet. Essence and Accidents of Software Engineering*, COMPUTER MAG. (Apr. 1987), *available at* http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html (last visited July 14, 2004).

29. *Id.*

processor's spell-checker has to work in 50 different languages, then it has to work in 50 different languages. In contrast, accidental complexity is created by programmers and can be dealt with.[30]

The accidental complexity of writing and optimizing machine code can be dealt with by programming in high-level languages that require fewer lines of code and have very strong checking routines that test the operation of module interfaces and help to minimize syntax and semantic errors. Similarly, using object-oriented programming methods helps to minimize the number of design inconsistencies. Note, however, that these changes only deal with the accidental complexity, not the essential complexity of the problems that we are trying to solve.[31]

Testing for errors is another strategy in preventing erroneous software from being released to an unsuspecting user.

D. How Is Software Tested?

Typically, during initial testing, software engineers exercise the instructions to see that they perform as expected. For example, if the first instruction is "copy the value 2 into register A," does register A contain the value 2 after the instruction's execution? Or, if you type a word into the word processor, do the letters appear on the screen? And, does the spell-checker highlight any misspelling?

Of course, programmers tend to test that their creation does what *they* expect it to do. If the original specification was wrong from a user's perspective, software can pass all its tests and still not meet the user's needs. Worse, engineers will often test in their environment, not acting as non-expert users. For example, knowing that the escape key will exit the debugger, they often do not test what happens when the end user presses the escape key.

During testing, some instruction paths may never be tested if the decision choosing that path is never exercised. In addition, within a single computer, different packages of software are likely to interact. Many a user has complained, "Everything worked until I installed xxxxx." Part of the problem is that these are very complex systems. Teams of individuals who may not see the whole picture write parts according to their interpretation of a specification they have received. All these parts must then be integrated into a whole containing

30. *Id.*
31. *Id.*

hundreds of different parts created by hundreds of people, all the parts coming together in one word processor or database.

There are literally hundreds of papers about software testing, ranging from the simplistic to the mathematically challenging. Most include the inherent assumption that it is not possible to remove all the errors from software.

E. Software Incidents—Deadly and Potentially So

The Association for Computing Machinery,[32] founded in 1947, maintains a "forum on the risks to the public in computers and related systems" called the *Risks Digest.*[33] It is a forum that provides a useful perspective on the risks associated with computers and software. Although not a complete database of known problems, a recent digest, Volume 23 with 45 issues covering the period from 7th November 2003 to 10th July 2004, contains just fewer than 680 entries of risk to the public. Items of interest range from the Nuclear Plant shut down by a lightning strike[34] (including a very short description of a computer failure causing the steam isolation valves to close), the driver of a Honda CRV trapped in a flood when his electrically driven windows would not wind down,[35] to the two accidents caused when the braking system of commuter buses was disabled by electromagnetic interference.[36]

There are other, deadly or potentially deadly examples. In February 2000, Ferrari issued a recall for its 360 Modena cars.[37] The reason for the recall was that "[t]he anti-lock braking system electronic control, under certain heavy braking conditions, may fail, resulting in the braking action being biased to the rear wheels, which

32. ACM: Ass'n for Computing Mach., Official Homepage, *at* http://www.acm.org/ (last visited July 14, 2004) ("[T]he world's first educational and scientific computing society").

33. ACM Comm. on Computers and Pub. Policy, THE RISKS DIG., *at* http://catless.ncl.ac.uk/Risks (last visited July 14, 2004).

34. ACM Comm. on Computers and Pub. Policy, *Nuclear Plan Shut Down by Lightning Strike*, 23 THE RISKS DIG. Issue 5 (Nov. 2003), *at* http://catless.ncl.ac.uk/Risks/23.05.html#subj3 (last visited July 14, 2004).

35. ACM Comm. on Computers and Pub. Policy, *Electronic Car Doors Trap Man in Australian Flood, Nearly Drown Him*, 23 THE RISKS DIG. Issue 6 (Dec. 2003), *at* http://catless.ncl.ac.uk/Risks/23.06.html#subj1 (last visited July 14, 2004).

36. ACM Comm. on Computers and Pub. Policy, *Loss of Bus Braking Due to Nearby Illegally Modified Transceivers*, 23 THE RISKS DIG. Issue 9 (Dec. 2003) *at* http://catless.ncl.ac.uk/Risks/23.09.html#subj2 (last visited April 14, 2005).

37. *All About Cars, Vehicle Recalls, Ferrari 360*, The AA, *at* http://www.theaa.com/allaboutcars/recalls/recalls.jsp?modelID=A5&modelName=360&makeName=Ferrari&makeId=B2 (last visited July 14, 2004).

may cause instability."[38] The repair was to replace the software in the anti-lock braking system electronic control unit.[39] Then, in April 2001, Ferrari issued another recall for the 360 Modena.[40] This time it reported that the brake failure warning light might fail to illuminate in the event of a system failure.[41] Again the repair was to update the software, but this time in the dashboard electronic control unit.[42]

Patients treated at the National Cancer Institute in Panama in November 2000 died after receiving an excessive dose of radiation from the Cobalt 60 radiotherapy machine.[43] An investigation determined that the cause of the overdose lay in the entering of data into the computerized treatment planning system.[44] The data could be entered in a number of different ways and when some particular methods were used, the output values were calculated incorrectly.[45] At the time of the initial radiological emergency notification, twenty-eight patients had been affected and eight had died.[46] The investigative team confirmed that five of the deaths were probably attributable to the patient's overexposure to radiation.[47] There was insufficient information to draw conclusions on two of the others, and the last one was considered to have died from his cancer.[48]

In Australia, software failure was found to have contributed to a fatal road accident. A baby boy was killed when a truck with defective brakes carrying compacted excavated clay crashed into the back of a car in which he was riding.[49] A glitch in the Roads and Traffic Authority software system had allowed the driver to reregister the truck without an inspection a month before the accident.[50]

In March 2003, a Royal Air Force Tornado[51] supersonic attack aircraft was struck by a missile fired by a USA "Patriot" air missile

---

38. *Id.*

39. *Id.*

40. *Id.*

41. *Id.*

42. *Id.*

43. *Radiological Accident at National Oncology Institute in Panama*, The Society for Radiological Protection, June 9, 2001, *at* http://www.srp-uk.org/servpanama.html.

44. *Id.*

45. *Id.*

46. *Id.*

47. *Id.*

48. *Id.*

49. Lorna Knowles, *Triple Truck Speed Led to Death of Baby Scott – RTA 'Glitch' Let Truckie Re-register Unsuspected Vehicle*, THE DAILY TELEGRAPH (Sydney), Feb. 25, 2003, at 9.

50. *Id.*

51. *Tornado GR4*, The Royal Air Force, *at*

defense system resulting in the deaths of two flight lieutenants.[52] In May 2004, the United Kingdom defense minister admitted that the Tornado software failed to identify itself as friendly and was then classified as an enemy rocket by the Patriot battery, which promptly shot it down.[53] An earlier BBC report had stated that the Patriot software had been identifying friendly aircraft as tactical ballistic missiles many times a day despite the fact that ballistic missiles travel at higher altitudes, on a predictable ascending or descending trajectory, and at higher speeds.[54]

When purchasing a computer it is common for the original hardware to come from one supplier, the firmware from a second supplier, and the installed operating system from the third supplier using configuration (system settings) from yet a fourth supplier. Add extra hardware, e.g., a Network card, from a second hardware supplier that alters firmware settings and uses drivers from yet another party. The drivers alter the way the operating system works and interacts with all other hardware; the new hardware itself interacts directly with the extant hardware. The hardware, firmware, and software were all designed, built, and tested by humans who are fallible. The original specification was created by a human as were the manuals for the end users. Failure at any stage can result in the aspirations of the user not being met. When dashed expectations also lead to injury and death, it is almost impossible for the injured party to pinpoint exactly what went wrong and who is responsible. Strict liability is seemingly appropriate for these very reasons.

III. THE LITERATURE ON SOFTWARE LIABILITY

Early commentators recognized the prospect of computers and software insinuating themselves into modern day life and the corresponding liability for damage caused by the new technology.[55] They pondered the circumstance of physical injury and whether strict liability would apply.[56] The drumbeat has not ceased. There are still ample examples in the literature of commentators considering the

---

http://www.raf.mod.uk/equipment/tornado_at.html (last visited Mar. 27, 2005).

52. *'System Error' Link to RAF Deaths*, BBC NEWS, May 14, 2004, *at* http://news.bbc.co.uk/1/hi/england/norfolk/3714251.stm (last visited Mar. 27, 2005).

53. *Id.*

54. *'Missile Error' Led to RAF Deaths*, BBC NEWS, Apr. 9, 2004, *at* http://news.bbc.co.uk/1/hi/uk/3613319.stm (last visited Mar. 27, 2005).

55. *See supra* note 1.

56. *See supra* note 1.

liability regime for defects in computer software and hardware.[57] Some of these commentators carefully track the emerging case law involving computers.[58] Not surprisingly, opinions among early and more recent commentators are split on whether strict liability is appropriate, or even necessary. Those who favor a strict liability regime generally do so based on the policies underlying the doctrine.[59] Those who oppose such a development cite concerns about stifling innovation,[60] cost,[61] stunting the growth of the industry,[62] and the possibility that beneficial products would become unavailable or would never be developed in the first place[63] if strict liability were to apply to software. We align with those who favor the adoption of strict liability. We do so for the policy reasons articulated by others, and also because we find that the industry is in a position to and should be made to absorb the cost of harm occasioned by defects.

IV. THE CASE LAW

The case law with respect to liability for software defects is large and growing. As always with the development of case law in an

---

57. Peter A. Alces, *W(h)ither Warranty: The B(l)oom of Products Liability Theory in Cases of Deficient Software Design*, 87 CAL. L. REV. 269 (1999); David W. Lannetti, *Toward a Revised Definition of "Product" Under the Restatement (Third) of Torts: Products Liability*, 55 BUS. LAW. 799 (2000); R.L. Mays, Jr., *Patent No. 6,035,321—Opening the Door to Software Product Liability Exposure*, 6 STAN. J. L. BUS. & FIN. 197 (2001); Douglas E. Phillips, *When Software Fails: Emerging Standards of Vendor Liability Under the Uniform Commercial Code*, 50 BUS. LAW. 151 (1994); Robert D. Sprague, *Software Products Liability: Has Its Time Arrived?*, 19 W. ST. U. L. REV. 137 (1991); J.P. Thurston, *Liability*, 4 COMPUTER L. & PRAC. 80 (1988); Wolpert, *supra* note 2; Brian H. Lamkin, Comment, *Medical Expert Systems and Publisher Liability: A Cross-Contextual Analysis*, 43 EMORY L.J. 731 (1994); Michael R. Maule, Comment, *Applying Strict Products Liability to Computer Software*, 27 TULSA L.J. 735 (1992); Patrick T. Miyaki, Comment, *Computer Software Defects: Should Computer Software Manufacturers Be Held Strictly Liable for Computer Software Defects?*, 8 SANTA CLARA COMPUTER & HIGH TECH. L.J. 121 (1992); Daniel T. Perlman, Note, *Who Pays the Price of Computer Software Failure?*, 24 RUTGERS COMPUTER & TECH. L.J. 383 (1998); Julia A. Tyde, Comment, *Medical Computer Software: Rx for Deadly Errors*, 4 SOFTWARE L.J. 117 (1990); Lori A. Weber, Note, *Bad Bytes: The Application of Strict Products Liability to Computer Software*, 66 ST. JOHN'S L. REV. 469 (1992).

58. Noriko Kawawa, *Comparative Studies on the Law of Tort Relating to Liability for Injury Caused by Information in Traditional and in Electronic Form: England and the United States*, 12 ALB. L.J. SCI. & TECH. 493 (2002); Sprague, *supra* note 57; Lamkin, *supra* note 57; Miyaki, *supra* note 57; Perlman, *supra* note 57; Weber, *supra* note 57.

59. Brannigan & Dayhoff, *supra* note 1; Gemignani, *supra* note 1; Sprague, *supra* note 57; Hall, *supra* note 1; Lanoue, *supra* note 1.

60. Lamkin, *supra* note 57; Miyaki, *supra* note 57.

61. Miyaki, *supra* note 57; Weber, *supra* note 57.

62. Freed, *Computer Age*, *supra* note 1.

63. Weber, *supra* note 57.

emerging field, the body of law is evolving slowly and incrementally. Also, as is typical, courts reason by analogy from known concepts and are applying historic principles to the more modern context of software.

Our reading of the cases throughout the years suggests eras into which the cases can be categorized. There are what we call precursor cases that do not involve software *per se*, but involve new or technology-based manufactured goods or simply set the stage for the upcoming software cases. On some level, all product liability cases could be considered precursor cases, but we confine ourselves to those that foretell the debate about strict liability for software. Next is the era in which the main issue is whether software is a good for purposes of the Uniform Commercial Code ("UCC") and a product for product liability generally. The intangible nature of software and the argument that it is really a service, not a good, dominate these cases. There is also a category of cases questioning whether a sale occurs when the buyer and seller identify their transaction as a license. Even when software is considered a tangible product and the transaction that transfers it is considered a sale, there are still questions about whether all liability regimes associated with the law of product liability—negligence, breach of warranty, and strict liability—should be brought to bear.

Early cases involved economic loss only as hardware and software malfunctioned and caused disappointment to the buyer, not physical injury. Because of the economic loss doctrine,[64] breach of warranty was considered the appropriate and only claim.[65] Within a breach of warranty regime, concepts such as disclaimers and limitations of warranties and remedies became important. Manufacturers and sellers quickly learned to include language in the contract of sale limiting their liability and damages for defects, as is permitted by the UCC.[66]

Even those cases that applied a negligence standard considered whether regular negligence principles should be applied or whether there should be a heightened standard of liability normally associated with professionals—a so-called programmer malpractice standard.

64. The economic loss doctrine is a judicially created principle that requires parties to live by their contracts rather than pursue tort actions for purely economic or commercial losses arising out of a contract. *See* Tietsworth v. Harley-Davidson, Inc., 677 N.W.2d 233, 241 (Wis. 2004). It is designed in part to maintain the fundamental distinction between tort and contract law. *Id.* at 242.

65. *See infra* notes 115–121 and accompanying text.

66. U.C.C. §§ 2-316, 2-719 (1972).

While this era is largely over, given the lack of professional standards in the industry, the question did occupy courts and commentators alike.[67]

Finally, there are the misrepresentation cases. In their enthusiasm to sell the new technology, sellers sometimes made assertions about the capabilities of their products that turned out not to be true.[68] When the promises were not fulfilled, disappointed buyers brought claims in misrepresentation. In so doing, they were often able to ignore the contractual limitations in the contract of sale that would have precluded or severely limited recovery.[69]

A. The Precursor Cases

The stage for software liability cases was set with *The T.J. Hooper* case.[70] The case involved a barge collision during a storm.[71] The tug boats towing the barges were not equipped with radios that could have alerted the captains to the fast-moving storm.[72] While radios were not required on tugs at that time, nor even widely adopted, the court nevertheless held that the lack of a radio was negligent:[73] "Courts must in the end say what is required; there are precautions so imperative that even their universal disregard will not excuse their omission."[74] *The T.J. Hooper* case is a technology-forcing case. The court was not willing to let industry custom set the standard of prudence when radio technology existed and was reasonably reliable and affordable,[75] although not widely adopted. The connection to software failure is clear. The software industry alone should not be permitted to set the standard for software quality lest it set the standard too low.

There are also the "book cases," a series of cases that held that information in a book is not a product for product liability purposes. These cases involve faulty information in books that, when acted

67. *See, e.g.*, Gemignani, *supra* note 1; Perlman, *supra* note 57.

68. *See, e.g.*, Clements Auto Co. v. Serv. Bureau Corp., 444 F.2d 169 (8th Cir. 1971); Dacotah Mktg. & Research, L.L.C. v. Versatility, Inc., 21 F. Supp. 2d 570 (E.D. Va. 1998); Accusystems, Inc. v. Honeywell Info. Sys., Inc., 580 F. Supp. 474 (S.D.N.Y 1984).

69. *See, e.g.*, *Clements Auto Co.*, 444 F.2d at 169; Caudill Seed & Warehouse Co. v. Prophet 21, Inc., 123 F. Supp. 2d 826 (E.D. Pa. 2000); *Accusystems, Inc.*, 580 F. Supp. at 474.

70. 60 F.2d 737 (2d Cir. 1932).

71. *Id.*

72. *Id.*

73. *Id.* at 740.

74. *Id.* at 740 (citations omitted).

75. *Id.*

upon by the reader, produced harm. Probably the most famous is *Winter v. G. P. Putnam's Sons*.[76] *The Encyclopedia of Mushrooms*, written by others and published by the defendant, contained erroneous information about the edibility of a particular mushroom.[77] Relying on the information, the plaintiffs ate what was actually a poisonous mushroom and became critically ill.[78] They sued the publisher under a number of theories, including strict liability.[79] The court held, among other things, that although the book itself was a product, the information in the book was not a product for product liability purposes.[80] However, in an interesting bit of dicta, the court imagines that "[c]omputer software that fails to yield the result for which it was designed" may be treated as a product,[81] and thus be subject to product liability law.

Other book cases include a chemistry text that misstated the steps of a chemistry experiment and caused injury to a student,[82] a recipe book that did not mention that an ingredient is toxic when eaten raw,[83] and a how-to book on tool construction that likewise produced injury.[84] In each of these "book cases," the courts refused to apply strict liability to the information in the book.[85] In addition to the reluctance to categorize information as a product, the courts were also concerned about First Amendment issues and chilling expression if they were to expose distributors of books to the heightened standard of strict liability.[86] The question then becomes whether information in a computer program is analogous to information in a book. We argue[87] that software is not information in the same way that the content of books are information because of its functionality and that

76. 938 F.2d 1033 (9th Cir. 1991).

77. *Id.* at 1034.

78. *Id.*

79. *Id.*

80. *Id.* at 1036.

81. *Id.*

82. Walter v. Bauer, 439 N.Y.S. 2d 821 (N.Y. App. Div. 1981).

83. Cardozo v. True, 342 So. 2d 1053, 1054 (Fla. Dist. Ct. App. 1977).

84. Alm v. Van Nostrand Reinhold, Co., 480 N.E.2d 1263 (Ill. App. Ct. 1985).

85. *Walter*, 439 N.Y.S. 2d at 822.

86. *See, e.g.*, Winter v. G. P. Putnam's Sons, 938 F.2d 1033, 1036–37 (9th Cir. 1991); *see also* Andrew T. Bayman, *Strict Liability for Defective Ideas in Publications*, 42 VAND. L. REV. 557 (1989); Jonathan B. Mintz, *Strict Liability for Commercial Intellect*, 41 CATH. U. L. REV. 617 (1992).

87. *See infra* notes 166–168 and accompanying text.

First Amendment concerns are not pressing when considering software as opposed to literature.[88]

In contrast to the book cases, but still within the precursor case category, are the "aeronautical charts" cases. The information on the charts can be analogized to the information in books. However, the courts that have considered the charts cases routinely hold that the charts are products for purposes of product liability and that strict liability can be applied. We detail these cases because we believe that they provide the best analogy to software and the fullest analysis and supporting arguments for why software should be subjected to strict liability. In *Aetna Casualty and Surety Co. v. Jeppesen & Co.*,[89] the Ninth Circuit upheld the District Court's finding that the charts in question were defectively designed.[90] The defendant, Jeppeson & Co., translated tabular data supplied by the Federal Aviation Administration ("FAA") and represented it graphically on charts used by pilots.[91] In *Aetna*, the flight crew was using two charts to undertake an instrument approach into the airport.[92] Unfortunately, the charts, showing different views of the approach, while accurate, were drawn to different scales and the plane crashed.[93] According to the court, "[t]he 'defect' in the chart consists of the fact that the graphic depiction of [one chart], which covers a distance of three miles from the airport, appears to be drawn to the same scale as the graphic depiction of [the other chart], which covers a distance of 15 miles."[94] The court did not engage in a thorough analysis of whether the chart was a product. Rather, it simply held that the difference in scale and the conflict created by the numbers on the charts and the graphic representation of those numbers "rendered the chart defective."[95] Indeed, the court's use of the *Restatement (Second) of Torts*, § 402A and case law that seemed to take on a strict liability approach,[96] makes for a reasonable case that the court adopted a strict product liability standard.

---

88. *See infra* notes 174–176 and accompanying text.
89. 642 F.2d 339 (9th Cir. 1981).
90. *Id.* at 342–43.
91. *Id.* at 341–42.
92. *Id.* at 342.
93. *Id.*
94. *Id.*
95. *Aetna Cas. and Sur. Co.*, 642 F.2d at 342.
96. *Id.* at 343.

In *Saloomey v. Jeppesen & Co.*,[97] a faulty chart was again responsible for an airline crash. The case went to the jury on all three theories of product liability: negligence, breach of warranty, and strict liability.[98] The jury ultimately found Jeppesen liable on all three theories and returned a verdict in favor of the plaintiffs.[99] In affirming the lower court the Second Circuit held:

> We believe that the trial court did not err in classifying appellant's charts as products. The charts, as produced by Jeppesen . . . reached [the pilot] without any individual tailoring or substantial change in contents—they were simply mass-produced. The comments to § 402A, *supra*, envision strict liability against sellers of such items in these circumstances. By publishing and selling the charts, Jeppesen undertook a special responsibility, as seller, to insure that consumers will not be injured by the use of the charts; Jeppesen is entitled—and encouraged—to treat the burden of accidental injury as a cost of production to be covered by liability insurance. This special responsibility lies upon Jeppesen in its role as designer, seller, and manufacturer.
>
> Appellant's position that its navigational charts provide no more than a service ignores the mass-production aspect of the charts. Though a "product" may not include mere provision of architectural design plans or any similar form of data supplied under individually-tailored service arrangements, the mass production and marketing of these charts requires Jeppesen to bear the costs of accidents that are proximately caused by defects in the charts.[100]

In *Brocklesby v. United States*,[101] the court took head-on the question of whether the aeronautical charts were products for purposes of strict liability. In this case the Jeppesen chart accurately portrayed an instrument approach procedure provided by the FAA.[102] The defects in the chart were a result of the FAA's faulty procedure.[103] Naturally, Jeppesen argued that it should not be held liable for a problem caused by the government.[104] The court pointed out that strict liability does not depend on fault and that, according to

97. 707 F.2d 671 (2d Cir. 1983).
98. *Id.* at 673.
99. *Id.* at 674.
100. *Id.* at 676–77 (citations omitted).
101. 767 F.2d 1288 (9th Cir. 1985).
102. *Id.* at 1295.
103. *Id.*
104. *Id.*

the *Restatement (Second) of Torts*,[105] sellers are strictly liable for injuries caused by a defective product even if they have exercised all possible care.[106] In response to Jeppesen's claim that it is unfair to hold the company strictly liable for republishing the government's information, the court said:

> Jeppesen's charts are more than just a republication of the text of the government's procedures. Jeppesen converts a government procedure from text into graphic form and represents that the chart contains all necessary information .... It is true that the government's procedures are significant components of Jeppesen's charts. It is apparent, however, that Jeppesen's charts are more than a mere republication of the government's procedures. Indeed, Jeppesen's charts are distinct products. As a manufacturer and marketer of those products Jeppesen assumed the responsibility for insuring that the charts are not unreasonably dangerous in their intended use.[107]

Finally, in *Fluor Corp. v. Jeppesen & Co.*,[108] a California state court reached a similar result as the previous cases. It cited with approval the previous aeronautical chart cases, saying that they were "entirely consistent with the fundamental policies which underlie the strict product liability doctrine in this state."[109] In considering whether a chart is a product for purposes of strict liability, the court held that "characterizing respondent's instrument approach charts as 'products' serves '[the] paramount policy to be promoted by the [doctrine],' i.e., 'the protection of otherwise defenseless victims of manufacturing defects and the spreading throughout society of the cost of compensating them.'"[110] The trial court had refused to give the jury a

---

105. RESTATEMENT (SECOND) OF TORTS § 402A (1965) provides:

> (1) One who sells any product in a defective condition unreasonably dangerous to the user or consumer or to his property is subject to liability for physical harm thereby caused to the ultimate user or consumer, or to his property, if
>
> (a) the seller is engaged in the business of selling such a product, and
>
> (b) it is expected to and does reach the user or consumer without substantial change in the condition in which it is sold.
>
> (2) The rule stated in Subsection (1) applies although
>
> (a) the seller has exercised all possible care in the preparation and sale of his product, and
>
> (b) the user or consumer has not bought the product from or entered into any contractual relation with the seller.

106. *Brocklesby*, 767 F.2d at 1296.

107. *Id.* at 1298.

108. 216 Cal. Rptr. 68 (Cal. Ct. App. 1985).

109. *Id.* at 70–71.

110. *Id.* at 71 (citing Campbell v. Gen. Motors Corp., 649 P.2d 224 (Cal. 1982)).

strict liability instruction because the chart was not a product[111] and that decision was reversed.[112]

The aeronautical charts cases provide a more perfect analogy to software than the book cases for a number of reasons. First, the charts are functional and not literary. There can be no purpose for aeronautical charts other than their function to aid pilots. Similarly, software has no purpose other than to cause a computer to perform some function. While it is true that the how-to books, such as recipe books, and mushroom hunting books are functional, they are a small subset of all books, which leads to the second argument. In the book cases, the courts expressed concerns that subjecting the information in the books to strict liability may chill free expression. In the case of aeronautical charts, free expression is not a desired quality. The information contained in the charts must be factual and accurate. There is no opportunity for literary license, political statement, or artistic expression. The same is true for software. The program must be accurate, and it must work. Indeed, in *Winter*, an early book case, the court opined that its holding that the book was a product but the information contained within the book was not, would not apply in the case of software.[113] The argument that subjecting information contained in books to strict liability has significant First Amendment implications is really a slippery slope argument. Even though the how-to books are functional, to hold in those instances that the information is a product sets the stage for holding all information in all books to that high standard. In other words, it is a short step to imposing that standard on all books, including literary and political tomes, and thus seriously implicating the First Amendment. A similar concern does not exist with respect to software. Like aeronautical charts, software is only functional and not literary or political. While it has been held that software enjoys some modicum of First Amendment protection, it is not the strict scrutiny standard that most literary expression enjoys.[114] Therefore, to subject defective software (i.e., software that does not function properly and causes injury) to strict liability does not risk opening up a universe of other software that is purely expressive to similar scrutiny as it does in the book cases. Therefore, we find that the aeronautical charts cases more

111. *Id.*

112. *Id.* at 75.

113. Winter v. G. P. Putnam's Sons, 938 F.2d 1033, 1036 (9th Cir. 1991).

114. *See* Universal City Studios, Inc. v. Corley, 273 F.3d 429, 449–52 (2d Cir. 2001).

nearly model the arguments for applying strict liability to defective software than do the book cases.

B. The Breach of Warranty Cases

Because almost all software cases to date involve purely economic loss, contract principles apply. Typically in these cases, sellers have attempted to limit their liability through contract either by disclaiming warranties, employing merger clauses, or limiting remedies.[115]

Courts have taken a number of approaches when faced with these cases. They may simply enforce the contract as written, complete with all its disclaimers, and deny recovery to the plaintiff. They may move outside the contract to find other bases for finding liability. Or, they may simply ignore the contract because it is unconscionable or is invalid for other reasons. Naturally, the approach taken depends heavily on the facts of each case and the existing case law in the jurisdiction.

Very early software cases involved large business systems that were typically custom-made, or at least customized, for the customer.[116] The contract was usually a product of face-to-face negotiations between the contracting parties and included training and service in addition to the system itself. In other words, the parties were often in a more equivalent bargaining position[117]—at least as to economic power, if not knowledge about the product—than one would find in a consumer contract. Nevertheless, even in the early cases, courts oftentimes strained to find a remedy that a strict reading of the contract would not seem to permit.[118]

---

115. *See* Phillips, *supra* note 57.

116. *See, e.g.*, Chatlos Sys., Inc. v. Nat'l Cash Register Corp., 635 F.2d 1081 (3d Cir. 1980); Sperry Rand Corp. v. Indus. Supply Corp., 337 F.2d 363 (5th Cir. 1964); Carl Beasley Ford, Inc. v. Burroughs Corp., 361 F. Supp. 325 (E.D. Pa. 1973); Sec. Leasing Co. v. Flinco, Inc., 461 P.2d 460 (Utah 1969); W.R. Weaver Co. v. Burroughs Corp., 580 S.W.2d 76 (Tex. App. 1979).

117. *See Chatlos Sys., Inc.*, 635 F.2d at 1087 ("[W]e find no great disparity in the parties' bargaining power or sophistication.").

118. *See Sperry Rand Corp.*, 337 F.2d at 371 (stating that integration clause in contract does not preclude establishing implied warranty of fitness for particular purpose); *Carl Beasley Ford*, 361 F. Supp. at 325 (rejecting the claim that written limitation on consequential damages does not overcome oral agreement for programming); Chatlos Sys., Inc. v. Nat'l Cash Register Corp., 479 F. Supp. 738 (D.N.J. 1979), *remedy modified*, 635 F.2d 1081 (3d Cir. 1980) (stating that additional damages may be appropriate even though written contract limits); *W.R. Weaver Co.*, 580 S.W.2d at 76 (finding the limitation of warranties to be valid, but stating that there may have been express warranties in statement of installation conditions). *But see Sec. Leasing Co.*, 461 P.2d at 460 (finding an integration clause effectively eliminated any chance to introduce

More modern warranty cases are similarly inconsistent. Some cases do not apparently involve any limitations and disclaimers and the courts simply award a contract remedy when the software fails to perform.[119] In those cases where a limitation provision is present, some cases uphold the contractual limitations and deny recovery.[120] There are also cases that find other ways for the purchaser to recover, even though a limitation provision would dictate a different result.[121]

Note that all the warranty cases rely on the UCC for their outcomes. Either tacitly or explicitly the courts have determined that software is in fact a good for UCC purposes[122] and the license agreement that transfers the software from buyer to seller is a

---

other evidence). It should be noted that these cases involve the U.C.C. and thus the courts were treating software as a good.

119. *See, e.g.*, Latham & Assoc., Inc. v. William Raveis Real Estate, Inc., No. 22 90 46, 1990 Conn. Super. LEXIS 688 (Conn. Super. Ct. May 10, 1990); Italo v. Monteleone, No. 83C-DE-70, 1986 Del. Super. LEXIS 1222 (Del. Super Ct. May 27, 1986); Winterbotham v. Computer Corps, Inc., 490 So. 2d 1282 (Fla. Dist. Ct. App. 1986); Carbur's, Inc. v. A & S Office Concepts, Inc., 445 A.2d 1109 (N.H. 1982).

120. *See, e.g.*, Lucre, Inc. v. ADC Telecomms., Inc., No. 1:02-CV-343, 2002 U.S. Dis. LEXIS 15421, at *14–15 (W.D. Mich. Aug. 16, 2002) (holding that the limitation of liability is not unconscionable); Brown v. SAP Am., Inc., No. 98-507-SLR, 1999 U.S. Dist. LEXIS 15525 (D. Del. Sept. 13, 1999) (holding that the software agreement limiting consequential damages is not unconscionable and thus controls); Ariz. Retail Sys., Inc. v. Software Link, Inc., 831 F. Supp. 759 (D. Ariz. 1993) (holding that the initial purchase of software is governed by licensing agreement that disclaimed warranties, although subsequent purchases do not indicate acceptance of original terms); Lyon Fin. Servs., Inc. v. Protech Plumbing & Heating, Inc., No. A03-810, 2004 Minn. App. LEXIS 191 (Minn. Ct. App. Mar. 2, 2004) (holding that there is a conspicuous disclaimer); Noble Thread Corp. v. Vormittag Assocs., Inc., 758 N.Y.S.2d 509 (N.Y. App. Div. 2003) (holding that the limitations are clear and unambiguous and not unconscionable); M.A. Mortenson Co. v. Timberline Software Corp., 998 P.2d 305 (Wash. 2000) (holding that the limitations in licensing agreement are not unconscionable).

121. *See, e.g.*, RRX Indus., Inc. v. Lab-Con, Inc., 772 F.2d 543 (9th Cir. 1985) (granting award of consequential damages even though the contract limited damages because the limitation failed in its essential purpose); Clements Auto Co. v. Serv. Bureau Corp., 444 F.2d 169 (8th Cir. 1971) (finding the warranty disclaimer to be valid but finding that the plaintiff had a cause of action for misrepresentation); Amsan, L.L.C. v. Prophet 21, Inc., No. 01-1950, 2001 U.S. Dist. LEXIS 16698 (E.D. Pa. Oct. 15, 2001) (finding that the limitation failed in its essential purpose); Caudill Seed & Warehouse Co. v. Prophet 21, Inc., 123 F. Supp. 2d 826 (E.D. Pa. 2000) (finding that the licensing agreement limits liability, but fraud claim is actionable); David Cooper, Inc. v. Contemporary Computer Sys., Inc., 846 S.W.2d 777 (Mo. Ct. App. 1993) (finding that the provision requiring return of software within 90 days was not an exclusive remedy and that the buyer had a reasonable amount of time to determine if goods were defective).

122. *See, e.g.*, Micro Data Base Sys., Inc. v. Dharma Sys., Inc., 148 F.3d 649 (7th Cir. 1998); Advent Sys. Ltd. v. Unisys Corp., 925 F.2d 670 (3rd Cir. 1991); RRX Indus., Inc. v. Lab-Con, Inc., 772 F.2d 543 (9th Cir. 1985); Softman Prod. Co. v. Adobe Sys., Inc., 171 F. Supp. 2d 1075 (C.D. Cal. 2001); Newcourt Fin. USA, Inc. v. FT Mortgage Co., 161 F. Supp. 2d 894 (N.D. Ill. 2001).

"sale."[123] Commentators have addressed in depth the issue of whether software is a good[124] and whether a license is a sale,[125] but the courts spend very little time, if any, debating the concepts. We will take their lead and similarly limit our discussion of these two points.

C. Cases Foretelling Strict Liability for Software

To date, there have been no reported cases holding a software manufacturer strictly liable for defects in the software. There are a few cases worthy of mentioning though because they may be harbingers of cases to come.

LaRossa v. Scientific Design Co.[126] is an early case refusing to apply strict liability to a supplier of radioactive pellets after employees who handled the pellets fell ill. Strict product liability was still in its infancy in 1968, and the reasons the court gave for not applying strict liability are instructive. The court pointed to three conditions that made strict liability inappropriate: the small market for the product, the fact that the market was highly specialized, and the fact that only a small group was affected.[127] Obviously the court was considering strict liability to be applicable in a mass-market context. It is curious to ponder whether the case would yield a similar result today after forty years of experience with strict liability and its policies.

In *General Motors Corp. v. Johnston,* an Alabama case,[128] a chip in a fuel delivery system of a Chevrolet truck was alleged to be faulty after it caused a fatal crash. The case was brought under the Alabama Extended Manufacturer's Liability Doctrine, and the jury awarded compensatory damages and punitive damages.[129] Although this was not a strict liability case, it is an example of the dire consequences and subsequent litigation that can ensue when software fails.

---

123. *See, e.g.*, I.Lan Sys., Inc. v. Netscout Serv. Level Corp., 183 F. Supp. 2d 328 (D. Mass. 2002); NMP Corp. v. Parametric Tech. Corp., 958 F. Supp. 1536 (N.D. Okla. 1997); ProCD, Inc. v. Zeidenberg, 908 F. Supp. 640 (W.D. Wis. 1996).

124. *See, e.g.*, Brannigan & Dayhoff, *supra* note 1; Deborah Kemp, *Mass Marketed Software: The Legality of the Form License Agreement*, 48 LA. L. REV. 87 (1987); Phillips, *supra* note 57; Horovitz, *supra* note 1; Lee Kissman, Comment, *Revised Article 2 and Mixed Goods/Information Transactions: Implications for Courts*, 44 SANTA CLARA L. REV. 561 (2004); Lanoue, *supra* note 1.

125. *See, e.g.*, Kemp, *supra* note 124.

126. 402 F.2d 937 (3d Cir. 1968).

127. *Id.* at 943.

128. Gen. Motors Corp. v. Johnston, 592 So. 2d 1054 (Ala. 1992).

129. *Id.* at 1056–57.

There is a second group of cases that never came to a full determination because they were settled. In one case, as told by one commentator, radiation patients received overdoses of radiation due to a software bug in the accelerator that administered the doses.[130] Two patients died and several others sustained serious injuries.[131] The estate of one of the deceased patients filed a lawsuit against the manufacturer of the accelerator and the cancer center where the patient received his treatments.[132] The complaint alleged that the product was defective and unreasonably dangerous and not fit for its intended use.[133] In other words, the claim was brought in strict liability. The case was eventually settled for an undisclosed amount of money.[134]

Another dispute arose when a device that allowed patients to self-administer pain medication after surgery apparently malfunctioned and caused a patient to overdose on Demerol.[135] The source of the malfunction was never formally determined because the case was settled, but it was presumed to be defective software.[136] The case was brought in strict liability,[137] but there is no formal outcome because of the settlement.

There are also news accounts of incidents that implicate faulty software, although the litigation has not come to trial.[138] In the fall of 2004, Medtronic, Inc. undertook a voluntary recall of software application cards for one of its medical products in the face of reports of patient injury and death.[139] This means that the stage is set for the courts to take head-on the question of whether the soft landing the software industry enjoys should come to an end.

---

130. *E.g.*, Tyde, *supra* note 57, at 136. Other commentators mention these cases, but Tyde provides the most complete narrative.

131. *Id.* at 137.

132. *Id.*

133. *Id.*

134. *Id.*

135. *Id.* at 137–38.

136. Tyde, *supra* note 57, at 138.

137. *Id.*

138. *See, e.g.*, *supra* notes 37–54 and accompanying text.

139. *See* Press Release, U.S. Food and Drug Administration, URGENT: Medtronic Announces Nationwide, Voluntary Recall of Model 8870 Software Application Card (Sept. 22, 2004), *available at* http://www.fda.gov/cdrh/recalls/recall-082404b-pressrelease.html. The company reported seven serious injuries and two deaths associated with the product. *See Medtronic: Company Announces Voluntary Recall of Software Application Card*, MED. & LAW WKLY., Oct. 22, 2004, at 191.

V. THE STRICT LIABILITY PARADIGM

The case of *Greenman v. Yuba Power Products, Inc*[140] is a case of great notoriety for its holding that "[a] manufacturer is strictly liable in tort when an article he places on the market, knowing that it is to be used without inspection for defects, proves to have a defect that causes injury to a human being."[141] The case is important because it ushered in a new era of product liability. No longer were plaintiffs put to the proof of actual negligence on the part of the manufacturer. No longer were contractual limitations on warranties and damages binding on the injured party. Strict liability placed the emphasis on the product itself and not on the conduct of the manufacturer and other players in the supply chain. The case signifies the recognition of what the consumer product market had become—mass-marketing of goods that were becoming ever more sophisticated and out of the ken of non-expert consumers; complicated chains of distribution that moved the product from producer to end user; a growing recognition that product-related injuries were heaping social costs onto the injured parties and the public at large.[142] The policy reasons underlying the doctrine are perhaps the most significant for purposes of this article. The *Greenman* court concluded that manufacturers were in the best position to prevent defects (and thus the injuries occasioned by them) and that the cost of the heightened liability could be spread across many products.[143] There is no question that strict liability ups the stakes for those who produce and sell products, but it is considered an appropriate measure of liability purely on public policy grounds.

The adoption of strict product liability was a quantum leap in product liability law. The conditions of its application include a defect in the product that causes injury to persons or property.[144] Its context is typically, but not limited to, mass-marketed goods,[145] a significant propensity for harm if the product is defective, an injured consumer who has no bargaining power to negotiate a deal with the manufacturer or seller different from the rest of the buying public, and a large disparity in knowledge about the product's technology

140. 377 P.2d 897 (Cal. 1963).

141. *Id.* at 900.

142. *Id.*

143. *Id.* at 901.

144. RESTATEMENT (SECOND) TORTS § 402A (1965).

145. By mass-marketed goods, we mean goods that are typically non-customized consumer products that are purchased off-the-shelf by consumers without expertise in the product's technology and with no or limited opportunity to inspect the product.

between the manufacturer and affected party.[146] These are precisely the conditions we find in today's software market. Consider, for example, the software-driven aspects of an automobile, which include the braking system, the fuel injection system, the electronic ignition system, and the airbags. A defect in any one of those software systems can easily result in significant physical injury or death. The software exists in all models of the automobile—the sale of which is governed by a standard contract. The consumer lacks knowledge about or is possibly unaware that software controls some aspects of the car's operation. The consequences, however, can be dire to the consumer if the software fails. Failure of the braking system or any of the engine management systems means that the driver loses control of the car. Air bags that fail to deploy or deploy when they should not cause injuries that would not have occurred but for the failure.

Admittedly, software embedded in medical equipment, airplanes, and air traffic control systems and used to monitor nuclear power plants is not mass-marketed in the same way that automobiles and other consumer products are, but the conditions that evoke strict liability remain the same. The consumer is unaware of and unknowledgeable about its workings. The consumer is not a party to the sale and consequently has no ability to negotiate the allocation of liability if something should go wrong. The likelihood of grave injury caused by a defect is significant. Thus, it would seem appropriate to apply strict liability to the context of software defects in these cases as well. There are some additional reasons specific to the software industry that compel the application of strict liability. First, to do so will remove the incentive to put "buggy software" on the market. There are those who claim that there is no such thing as "bug-free software."[147] They argue that the sheer complexity of the product, which frequently contains millions of lines of code, means that no amount of testing can determine every permutation of the program and whether it will operate as planned in all combinations.[148] Critics nevertheless believe that software is sometimes rushed to the market to gain competitive advantage and in the hope that users will find the bugs and report them so they can be corrected in subsequent releases of the product.[149] In other words, the public release of the product is,

146. LaRossa v. Scientific Design Co., 402 F.2d 937 (3d Cir. 1968).

147. Freed, *Computer Age*, *supra* note 1, at 275; Gemignani, *supra* note 1, at 195.

148. *See, e.g.*, Gemignani, *supra* note 1 at 185.

149. *See* Kevin Coughlin, *Math Wizards Ask What's Bugging Computers*, NEWHOUSE NEWS SERV., June 9, 2004.

in effect, one big beta test of its capabilities. Presumably software that can cause serious injury and death is not marketed so cavalierly. However, it is indicative of what can happen when manufacturers are held to a mere negligence standard of liability. The basic premise that software inevitably contains bugs dictates the application of strict liability for two reasons.

First, when tragic consequences occur, the doctrine relieves the plaintiff from having to prove negligence. In a program that contains millions of lines of code, it may be that the presence of a few bugs will not qualify as negligence and relieve the producer of liability for any injuries. On the other hand, a strict liability standard increases the liability *ex ante* and will give the producer pause when selecting and running tests on the software. Perhaps a few tests are not enough if more testing can tease out the defects that lie latent in the program. That is precisely the idea behind strict liability: Increase the incentives to manufacture and distribute a product without defects that cause harm.

Second, software manufacturers gain a benefit from putting their products into the stream of commerce and employ intellectual property concepts to assert a monopoly over their wares. Software can be protected by patent[150] and copyright,[151] and the assertion of these rights means that the manufacturer is the sole market participant for that particular product. While other product manufacturers can and do use primarily patent law to protect their products, the situation is slightly different. Many consumer products do not depend on a notion of compatibility. One co-worker can own a Ford and another may own a Toyota, without any implications whatsoever. Those same two co-workers, however, will need their office software to be compatible in order to do their jobs. Only Microsoft produces Windows and, if that is the platform the office supports, everyone in the office will have to be a Windows user or risk not having the appropriate and necessary technical support. More related to our situation is the following situation. If a hospital employs a medical monitoring system in the intensive care unit, every patient in the unit will be monitored by the same system. It would be chaos to have multiple systems employed in the same hospital. Compatibility and familiarity with one system are essential to quality care. Even if competitors create similar non-infringing products, they will not easily find a market where another system is in place. Thus, the monopoly created

150. Diamond v. Diehr, 450 U.S. 175 (1981).

151. 17 U.S.C. § 102 (2000).

by intellectual property laws combined with the high compatibility need in most settings virtually assures that software manufacturers can stave off competition and benefit from their rarified position in the market in ways their counterparts in other industries cannot. Strict liability for defective products seems to be a small price to pay for the benefits reaped in such a marketplace.

We are not unmindful of the arguments against strict liability generally and as it applies to software in particular. Of course there are increased costs that accompany increased liability. These are costs associated with enhanced testing, defending a claim for damages, the actual payment of damages or settlement, and the cost of liability insurance. These are costs that are borne by all other manufacturers who then calculate those costs into the price of the product. It is anomalous to segregate software for different treatment when that industry shares similar characteristics with those manufacturers that bear strict liability. The products of the software industry have as much capacity to do harm as the products of all other industries, yet they do not face the same liability exposure. If one considers that the cost of software—particularly mass-marketed software—is going down and not up,[152] it does not seem unduly burdensome to impose these added costs on the manufacturer, who, as *Greenman* noted over forty years ago, has the ability to build the increased cost into the price.[153]

We are unmoved by the argument that imposing strict liability will stifle innovation, especially because we are focusing on a segment of the industry—software that foreseeably causes physical harm when defective—rather than the entire software industry. That has certainly not been the case in other industries subject to strict liability standards. We know of no special characteristic of the software industry that would cause a stifling effect to happen there. In fact, Professor Samuelson characterizes the software industry as one of rapid innovation and strong competition.[154] She says it is evidence that the software market is vibrant and successful.[155] Those are precisely the reasons that the stifling effect—if there is one at all—will not occur. We contend that a vibrant and successful market characterized by strong competition will be able to adjust to the new

152. Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308, 2376 (1994).

153. Greenman v. Yuba Power Prods., Inc., 377 P.2d 897, 901 (Cal. 1963).

154. Samuelson et al., *supra* note 152, at 2378.

155. *Id.*

liability regime. The industry has the resources and the depth of experience to continue innovating in that environment because the market will continue to reward innovation, even in the face of strict liability, as it has in other industries.[156]

These same conditions vitiate the argument that the imposition of strict liability would stunt the growth of the industry.[157] The software industry is no longer in its infancy. It has grown by leaps and bounds and is now a major player in the economy. All the conditions are in place that will allow continued growth of the industry. Software is ubiquitous in commercial and consumer life as public acceptance of and need for software products increase. The market will surely demand that more software applications be invented and commercialized. We express no opinion about whether the imposition of strict liability would have stunted the growth of the industry in its infancy. However, we are not persuaded that such a move will have a similar effect now that the industry is such an important component of the economy. We return again to the point that a similar fate has not been visited on other product manufacturers whose products foreseeably cause harm when defective, and we see no evidence to indicate that the software industry differs in any way that would cause a different result.

More troubling perhaps is the argument that fears of heavy liability exposure will keep beneficial products out of the market.[158] We are unaware of any specific instances of software not being developed or non-defective software being pulled from the market because of this fear; however, we know the allegation has been made with respect to other products such as vaccines and aircraft.[159] If software can help relieve suffering and enhance the quality of life, we would want to create an environment where its development can flourish. By the same token, if the harm that defective software causes outweighs the benefits sought, it is right and fitting that it be driven from the market. We trust the case law to make that determination. Courts have long and rich experience with considering the social benefits of products and conduct and calibrating the incentives where

156. *See, e.g.*, Maria Papadakis et al., *Strict Liability and Consumer Product Innovation: Results from a Cross-industry Pilot Study*, 12 INTERN'L J. TECH. MGMT. 324 (1996) (finding that the introduction of a strict liability regime in the European Union was a factor but not a prime motivator of innovation and was a positive influence on some types of innovation).

157. *See* Freed, *Computer Age*, *supra* note 1.

158. *See* Weber *supra* note 57.

159. Frances E. Zollers et al., *Looking Backward, Looking Forward: Reflections on Twenty Years of Product Liability Reform*, 50 SYRACUSE L. REV. 1019, 1028–32 (2000).

social benefit will be maximized. Moreover, courts' experience with crafting the contours of strict liability is similarly extensive. Consequently, we favor an approach that accepts strict product liability for software and lets the case law find the point of maximum social benefit through the application of a rigorous cost/benefit analysis. In the rare case where a beneficial software product is being kept out of the market because of liability concerns and that claim is corroborated, there are a number of public policy strategies that can be employed in mitigation. For example, there is federal legislation limiting vaccine manufacturers' liability for certain vaccines.[160] The legislation was thought necessary after manufacturers threatened to withdraw life-saving vaccines from the market in the wake of product liability verdicts.[161] More recently, a federal law has been passed to relieve manufacturers who produce anti-terrorism products and processes from liability for damages associated therewith.[162] While we do not favor a legislative approach for reasons that will be detailed below,[163] it is nevertheless an option if beneficial software is truly not being brought to market because of liability concerns. Bear in mind that strict liability applies to *defective* products; we wish to create an environment where beneficial and not defective software can be marketed, but where defective software is discouraged in the first instance and its producers are made liable for the defects that cause injury.

When reviewing the costs and benefits of applying strict liability to defective software, we strike the balance in favor of its application. As the *Greenman* court said so many years ago, it is sound policy to place the burden on the one who is in the best position to avoid the defect in the first place,[164] even though doing so may pose an extreme hardship on one producer or another.

VI. WHY SOFTWARE? WHY NOW?

The previous section discussed the policies underlying strict liability and how those policies are fostered by applying the doctrine to software. In this section we consider the conditions under which strict liability would be so applied and the limitations on doing so.

160. National Vaccine Injury Compensation Program, 42 U.S.C. § 300aa-10 (2000).

161. *Id.*

162. Support Anti-terrorism by Fostering Effective Technologies Act of 2002 (the SAFETY Act), 6 U.S.C. § 441 (Supplement 2004).

163. *See infra* notes 196–199 and accompanying text.

164. Greenman v. Yuba Power Prods., Inc., 377 P.2d 897, 901 (Cal. 1963).

Additionally, we articulate specific attributes of the software industry that suggest the timing is right. In so doing, we rely heavily on the aeronautical charts cases for support.[165]

First, we argue that the software should be considered a "product" for product liability purposes. We find wholly unpersuasive the claim that software is information only and is intangible and should not be considered in the same category as automobiles, pharmaceuticals, and even hardware. Software's value lies in the fact that it does something, i.e., it performs a task.[166] Although Professor Samuelson was not writing about product liability, she asserts that computer programs and physical machines are more alike than different.[167] For example, like machines, each unit performs an identical task, is comprised of many component parts (lines of code), and the components must work together like a machine with gears and pulleys.[168] Cases have held[169] and commentators have argued[170] that software is a good for both warranty and strict liability purposes. Rather than trying to make software conform to a definition that would be appropriate for a physical good, we think the better approach is to say that software will be treated as a good, whether it actually is a good or not. In other words, it shares enough characteristics of a good that, when viewed in the light of strict liability policies, it should similarly be considered a good. Electricity has been considered a good,[171] as have blood[172] and aeronautical charts.[173] Software should be no different than those examples because of its use by people, its capacity to do harm, and the inability of those affected to meaningfully evaluate its safety.

---

165. *See supra* notes 89–112 and accompanying text.

166. Samuelson et. al., *supra* note 152, at 2316.

167. *Id.* at 2320.

168. *Id.* at 2320–22.

169. *See supra* notes 122–123 and accompanying text.

170. *See* Kawawa, *supra* note 58, at 498; Phillips, *supra* note 57; Horovitz, *supra* note 1.

171. *See, e.g.*, Van Hoose v. Blueflame Gas, Inc., 642 P.2d 36 (Colo. Ct. App. 1981), *aff'd*, 679 P.2d 579 (Colo. 1984); Elgin Airport Inn, Inc. v. Commonwealth Edison Co., 410 N.E.2d 620 (Ill. App. Ct. 1980), *rev'd in part*, 432 N.E.2d 259 (Ill. 1982); Schriner v. Pa. Power & Light Co., 501 A.2d 1128 (Pa. Super. Ct. 1985); Ransome v. Wis. Elec. Power Co., 275 N.W.2d 641 (Wis. 1979).

172. *See, e.g.*, Cunningham v. McNeal Mem'l Hosp., 266 N.E.2d 897 (Ill. 1970); Jackson v. Muhlenberg Hosp., 232 A.2d 879 (N.J. Super. Ct. Law Div. 1967), *rev'd*, 249 A.2d 65 (N.J. 1969). *But see, e.g.*, Shepard v. Alexian Bros. Hosp., Inc., 109 Cal. Rptr. 132 (Cal. Ct. App. 1973); Samson v. Greenville Hosp. Sys., 377 S.E.2d 311 (S.C. 1989).

173. *See supra* notes 89–112 and accompanying text.

To declare software a product similarly does not implicate the First Amendment. The book cases[174] did not treat information contained within the books as products, in part, because of concerns about chilling expression. Software is not information in the same way that the words in a book are information. Returning to the functionality argument, software's value lies within its behavior, not its text.[175] It is embedded in devices to cause them to work. It is not the expression that is important, but rather its function. Correspondingly, the information in software code is factual[176] (like aeronautical charts)—not editorial, not political, not literary—and, as such, it is not in need of the full measure of First Amendment protection that other kinds of information enjoy.

The Restatement (Third) of Torts contains a definition of product that some say[177] would preclude the inclusion of software in it.[178] To the extent that it may be true, we urge courts not to adopt the definition and, instead, focus on the policies underlying strict liability, not verbal niceties, when applying strict liability to software. We align with the view of David Lannetti that product liability law must be flexible enough to include "new items that may not qualify as products under current guidance but nevertheless should be governed by existing product liability law, based on the policy objectives undergirding this field of law."[179] Courts and commentators have routinely considered software a good under the UCC.[180] It does not strain credulity to transition from "good" to "product" for strict liability purposes and based on policy grounds, regardless of the *Restatement*'s language.

---

174. *See supra* notes 76–86 and accompanying text.

175. Samuelson, *supra* note 152, at 2315.

176. *See* Lamkin, *supra* note 57, at 767.

177. *See* Lannetti, *supra* note 57.

178. For purposes of this Restatement:

> (a) A product is tangible personal property distributed commercially for use or consumption. Other items, such as real property and electricity, are products when the context of their distribution and use is sufficiently analogous to the distribution and use of tangible personal property that it is appropriate to apply the rules stated in this Restatement.
>
> (b) Services, even when provided commercially, are not products.
>
> (c) Human blood and human tissue, even when provided commercially, are not subject to the rules of this Restatement.

RESTATEMENT (THIRD) OF TORTS § 19 (1998).

179. Lannetti, *supra* note 57, at 800 n.9.

180. *See supra* notes 122–125 and accompanying text.

Similarly, the question of whether software is actually sold when it is licensed does not concern us.[181] Courts have routinely applied UCC principles to software transactions[182] without splitting hairs over whether the transfer is a license—as characterized by the parties—or actually a sale. The transfer puts the product into the stream of commerce where it can cause harm. That fact alone should dictate strict liability, regardless of how the parties label their transaction.

Having dispensed with the license/sale and goods/intangibles dilemmas, we turn to the conditions under which we envision the application of strict liability to software. The first component must be an injured user, consumer, or bystander. This will effectively eliminate all the economic loss cases from the broad sweep of strict liability. Many recent articles undertake an analysis of who should be liable for insecure systems that permit hackers, the introduction of viruses, terrorists, and the like.[183] Unless these instances give rise to foreseeable physical injury, they would not be treated under a strict liability regime according to our conception. Even then, it would be necessary for a defect to cause the harm, not an outside force misusing the software. Also, the myriad of cases that are brought by purchasers who want their money back or their business losses compensated would not be able to utilize this cause of action. We confine our call for strict liability to cases where there is a foreseeable risk of physical injury to people when software is defective.

Next, we envision a mass-marketed product. Most custom software is too bound up in the product/service dichotomy to qualify. That being said, we define mass-market very liberally. We are mindful of the aeronautical charts cases where the courts held that the chart was a product, in part, because it was mass-marketed.[184] The charts are for use by pilots—a very small segment of the population. They are not available in the local bookstore or consumer goods store. Yet, as with the computer software we are considering, if there is a defect in the chart, the consequences extend well beyond the user of the chart. Consider medical monitoring software. It is not available to

181. *See, e.g.*, Kemp, *supra* note 124 and cases cited note 122.

182. *See supra* note 123 and accompanying text.

183. *See, e.g.*, Robin A. Brooks, *Deterring the Spread of Viruses Online: Can Tort Law Tighten the 'Net'?*, 17 REV. LITIG. 343 (1998); Cheryl S. Massingale & A. Faye Borthick, *Risk Allocation for Computer System Security Breaches: Potential Liability for Providers of Computer Services*, 12 W. NEW ENG. L. REV. 167 (1990); Michael D. Scott, *Liability for Insecure Systems*, 9 CYBERSPACE LAW. 1 (Mar./Apr. 2004); Steve Lohr, *Product Liability Lawsuits Are New Threat to Microsoft*, N.Y. TIMES, Oct. 6, 2003, at C2.

184. Saloomey v. Jeppesen & Co., 707 F.2d 671 (2d Cir. 1983).

just any purchaser, but the impact, should it be defective, is wide and devastating to those who are harmed by the software, even though they did not purchase it and it was not marketed to them. The air traffic control systems are guided by software. Although the market for the actual system is very limited, the number of persons affected by the system is enormous. Software in automobiles, on the other hand, fits the usual mass-marketing concept. Many people purchase automobiles made by a number of manufacturers. While millions of automobiles are purchased every year, the impact of defective software within the automobile is not much different from that of a defective air traffic control system, which is sold to a very limited group of purchasers. In defining mass-market, we do not evaluate the number of units sold, but rather the scope of the impact a defect in the software would have on people. "Mass-marketed" must be distinguished from "marketed to the masses." If the software has the potential to affect in harmful ways a large number of people,[185] regardless of how widely it is distributed, it meets this standard.

Harkening back to *Greenman v. Yuba Power Products, Inc.*[186] we would require that the consumer be unable to evaluate the quality of the product so that the burden falls to the manufacturer to make a quality product in the first instance. In this age of technology, this is an easy requirement to meet. Neither the passenger aboard a commercial flight, the patient in the hospital, nor even the driver of a car can understand or even see the software operations going on around him or her. Very few in the general population can read, let alone understand, the object code that comprises software.

Most importantly, and as discussed earlier,[187] we believe that the industry is sufficiently mature to bear the cost of strict liability. It is generally understood that early tort law developed to protect emerging industries.[188] That may have been the case with software as well, but it is no longer accurate to refer to the industry as emerging

185. See LaRossa v. Scientific Design Co., 402 F.2d 937 (3d Cir. 1968), where the small group affected by the product compelled the court not to apply strict liability.

186. 377 P.2d 897 (Cal. 1963).

187. *See supra* notes 154–157 and accompanying text.

188. *See, e.g.*, Michael Rustad & Lori E. Eisenschmidt, *The Commercial Law of Internet Security*, 10 HIGH TECH. L.J. 213, 259–62 (1995) (stating the defenses in tort law developed to protect industrialization, citing MORTON J. HORWITZ, THE TRANSFORMATION OF AMERICAN LAW 1780–1860, 99–161(1977)); Michael L. Rustad & Thomas H. Koenig, *Cybertorts and Legal Lag: An Empirical Analysis*, 13 S. CAL. INTERDISC. L.J. 77 (2003) (finding the information industry to not be paying its way, just like railroads, canals, and factories in 19th century); Wolpert, *supra* note 2, at 519 (stating that the liability for defective software has been slow to develop).

and in need of protection. Previously we described the growth of the industry.[189] Today, the industry is a multi-billion dollar segment of the economy. The market has seen significant consolidation and now a few companies dominate in the major product categories.[190] A 2002 study by the National Institute of Standards and Technology pegged the cost of faulty software at $59.5 billion a year.[191] All of these indicators say that the industry is capable of paying for the cost of injuries due to defects in the software. Other product manufacturers have managed the liability issue. Software manufacturers should as well. Otherwise, the industry is relieved of paying the true cost of its wrongdoing.[192] That is precisely what tort law seeks to avoid.

We recognize that the application of strict liability to defective software includes all the concepts, defenses, and limitations that accompany its application to all other products. We review some of them next. Because strict liability is a creature of the common law, there is some variation from state to state, exacerbated by statutory limitations that a number of states have enacted.[193] We focus on elements of the strict liability claim that are common to most states.

A. Manufacturing or Design Defect?

Unless there is an error in the copying of software code, software defects are likely to be considered design defects. As stated previously, software that does something unexpected is nevertheless responding exactly as it has been programmed to do. *The Restatement (Second) of Torts* applies a consumer expectation test to design defects,[194] i.e., the product was defective if it did not meet the reasonable consumer's expectations of safety. In contrast, the *Restatement (Third)* returns design defect cases to a negligence standard when it requires that, in order for there to be liability, the "foreseeable risks of harm posed by the product could have been reduced or avoided by the adoption of a reasonable alternative . . . and the omission of the alternative design renders the product not reasonably safe." [195] The requirement of a *reasonable* alternative design is reminiscent of the negligence standard. While we prefer that

189. *See supra* notes 3–31 and accompanying text.

190. Samuelson et al., *supra* note 152, at 2376.

191. *See* Coughlin, *supra* note 149. The number includes costs such as fixes, downtime, and lost business and is not confined to the costs of physical injury.

192. Rustad & Koenig, *supra* note 188, at 140.

193. Zollers et al., *supra* note 159, at 1032–40.

194. RESTATEMENT (SECOND) OF TORTS § 402A (1965).

195. RESTATEMENT (THIRD) OF TORTS § 2(b) (1998).

design and manufacturing defects both be tested under a pure strict liability theory, even those states that have adopted the *Restatement (Third)* can find liability for faulty software design.

The reasonable alternative design standard assumes that the technology has not advanced to the point, or that the inherent characteristics of the product are such that the danger can be avoided. An example would be a pharmaceutical product that does what it is supposed to do, but the biochemistry, even in its perfect state, produces serious side effects. In the case of software, the reasonable alternative design is the one that does what the software is supposed to do and not something else. The product is marketed to, and usually does, produce a particular result. Thus, medical software is designed to monitor accurately the patient's condition and airplane software is designed to fly a plane safely. When a bug in the software causes injury or death, there is a reasonable alternative design—the program without the bug that works properly.

B. Causation

Naturally, the defective software must be the cause of the eventual harm. Claims can be made that the operator, not the software, was the causative agent in producing the harm. This was true in many of the aeronautical charts cases, where the chart maker claimed pilot error. One explanation, later refuted, in the case of over-radiation was that the technician did not operate the machine correctly. Years of litigation against Audi for the sudden acceleration of some of its cars finally ended with the conclusion that drivers were inadvertently hitting the gas instead of the brakes, thereby causing the cars to lurch forward.

Our concept of strict liability applied to software requires that the injured party establish by a preponderance of the evidence that a defect in the software caused the harm. Sometimes this will not be easy to establish. However, the element of causation is absolutely essential to the cause of action. Without it, software manufacturers could be liable for anything that happens in proximity to the software. That is not our intent; rather, we propose that manufacturers be responsible for *defects* in their product that *cause* injury.

C. State of the Art Defense

Either by case law or by statute, many states recognize the state of the art defense in a strict liability claim, i.e., if the product conforms to the state of the art at the time it is produced, then it will

not be considered defective. It could be argued that the state of the art in software is that all products contain bugs, and it is impossible to produce a software program without them. We find this argument wholly unpersuasive.

Software has matured to the point where it can and does operate safely. Millions of people travel safely every day to and from their destinations in automobiles loaded with software. Legions of patients are monitored accurately and receive appropriate treatment with medical software. Millions of airline passengers fly safely in planes controlled by software. The state of the art is such that software that performs properly and safely is a reality, not a dream. Consequently, the state of the software art and technology has evolved to a point where we can expect it to do what it is supposed to do. Any deviation from that should not be sheltered by the state of the art defense.

The defense has the effect of turning strict liability back to a negligence standard. While it may be true that failure to detect an error in millions of lines of code is not negligence that is not what the strict liability doctrine is about. Strict liability holds manufacturers of products to a higher standard in order to ensure that safe products reach the market. If the doctrine dissuades software manufacturers from releasing products that can cause serious injury and death when defective, then it has achieved its purpose.

VII. CONCLUSION

It may seem an odd time to be advocating for an extension of the strict liability doctrine to include software. The current product liability environment is one of restriction, not extension. Tort reform legislation that restricts some pieces of established product liability doctrine is occurring in the states,[196] and, in a piecemeal way, at the federal level.[197] With respect to software specifically, the Uniform Computer Information Transaction Act[198] (UCITA), specifically

196. Zollers et al., *supra* note 159, at 1032–40.

197. While comprehensive federal reform has not occurred, individual pieces of legislation have limited liability for particular industries and in particular settings. Examples include vaccines (National Vaccine Injury Compensation Program, 42 U.S.C. § 300aa-10 *et. seq.* (2000)), private aircraft (General Aviation Revitalization Act of 1994, 49 U.S.C. § 40101 (2000)), industries creating anti-terrorism products (Support Anti-terrorism by Fostering Effective Technologies Act of 2002, 6 U.S.C. § 441 *et. seq.* (Supplement 2004)), and the recent limitations on class action lawsuits (Class Action Fairness Act of 2005, Pub. L. No. 109-2, 119 Stat. 4 (2005)).

198. UNIF. COMPUTER INFO. TRANSACTION ACT §§ 101 *et. seq.* (2002). The Act has only been adopted by two states, Maryland (Maryland Uniform Computer Information Transactions Act, MD. CODE ANN., COM. LAW II §§ 22-101 to -816 (Supplement 2003)) and Virginia

allows disclaimers and limitations of liability in electronic transfers of software and information. However, none of these constricting developments dissuade us from our position that the time is right for software to be subjected to strict liability.

In the forty-plus years that strict liability has been a part of the legal landscape, manufacturers and suppliers of goods have adjusted their manufacturing, testing, and inspection processes to adjust to the heightened liability. It may be true that some businesses were not able to cope with the legal environment and failed, but the market place has rewarded those who were able to respond appropriately. We submit that it will be no different with software. Granted, certain practices will have to be strengthened or abandoned altogether, but we submit that is precisely the point. If the end result is to put improved software on the market that works properly and does not cause injury to people, then the doctrine will have accomplished its purposes.

We rely on the courts, not legislatures, to make the transition. In their historic role of crafting a common law that reflects sound public policy, the courts are in the best position to create this new environment. Strict liability is a creature of the common law; thus, it is appropriate for courts to continue to refine and develop the doctrine as changes in technology occur. The seeds to grow the doctrine are present in existing case law, whether or not the case is about software. It is actually a small step to move from applying strict liability to cases involving aeronautical charts or car braking systems that are not driven by software to ones that are. Most important, courts are more removed than legislative bodies from the political exigencies that currently call for the contraction of liability rules. While courts often experience a so-called legal lag[199] when addressing new technologies and new social conditions, we believe that the lag period is over. The software industry has enjoyed a half-century honeymoon period in which it could develop and flourish. It is now time for it to contribute to the costs society incurs from software's use. We believe that the courts are in the best position to recognize that fact and make it happen.

(Uniform Computer Information Transactions Act, VA. CODE ANN. §§ 59.1-501.1 to -509.2 (Michie 2001)). Both states have a substantial high-technology industry presence. Other states have passed so-called bomb shelter legislation, providing that the disclaimer provisions of UCITA will not become law in those states. *See* H.F. 2205, 78th Gen. Assem., Reg. Sess. (Iowa 2000); S.B. 1023, 2001 Leg., 144th Sess. (N.C. 2001); S.B. 204, 75th Gen. Assem., 1st Sess. (W. Va. 2001); H.B. 148, 2003 Leg., 67th Biennial Sess. (Vt. 2003).

199. *See, e.g.*, Rustad & Koenig, *supra* note 188.

As we review the existing cases and the news accounts of software failures that cause personal injury, we are convinced that the timing is right for the shift to strict product liability. There have already been instances of software failure causing injury, and there can only be more as software interfaces with ever-increasing amounts of human activity. Given the current uses of software, it is not hard to imagine future applications and the drastic consequences that will result if the software is faulty. We think it sound public policy to have a strict liability regime in place to treat those consequences, should they occur, rather than putting injured parties to the task of proving negligence.

Whatever space the software industry may have needed to develop and mature has surely come and gone. The industry is consolidating and strengthening, and large market players have emerged as dominant forces in the economy. Realistically, it is not the independent software developer or a small company that develops the software we seek to hold strictly liable should something go wrong. The sophistication and complexity required of medical monitoring software and the software that runs power plants and planes is typically developed by established companies. These companies not only have the technological ability to create the software, but also the resources to devote to the extra testing that strict liability requires. They also have the resources to pay for injuries that may result from defects or to purchase insurance to cover such an eventuality.

All the conditions are in place for the application of strict liability to software defects. The case law is poised to move in that direction. As software that can cause injury if defective becomes more and more a part of our daily lives, the policy reasons underlying strict liability are congruent with the application of the doctrine to software. The industry is sufficiently established and mature to be able to withstand the enhanced liability. The reasons for treating software differently from all other products no longer withstand close scrutiny, if they ever did. We urge the extension of strict liability to defective software that causes injury at the earliest opportunity.